Facultad
de
Ciencias

Marco para el desarrollo de aplicaciones Ada con sensores/actuadores Lego en Arduino

Framework to develop Ada apps with Lego sensors/actuators in Arduino

Trabajo de Fin de Grado
para acceder al
Grado en Ingeniería Informática

Autor: Daniel Fernández Castillo
Director: Héctor Pérez Tijero
Co-Director: Mario Aldea Rivas
Julio 2021

Resumen.

El progreso tecnológico ha permitido la globalización de la robótica en todos los aspectos de nuestra vida. Hoy en día es un ámbito ampliamente utilizado tanto a nivel personal, como industrial y últimamente con una gran importancia en el aspecto académico.

Es el interés en el avance de la robótica el que ha motivado a la realización de este proyecto, teniendo como objetivo elaborar una plataforma de desarrollo de aplicaciones Ada para sensores y actuadores LEGO. Esta plataforma estará enfocada hacia un nivel educativo, permitiendo de ese modo su uso a docentes y alumnos en futuras prácticas de aplicaciones concurrentes del grado de informática de la Universidad de Cantabria.

LEGO, en colaboración con ROBOTIX, dispone de una linea de robótica denominada Mindstorm. Dicha plataforma dispone de numerosos sensores y actuadores con los que se pueden realizar una amplia variedad de proyectos. Además, otra de la característica que presenta esta serie es la facilidad de uso así como el precio ajustado de sus dispositivos, permitiendo su alcance a todo tipo de público.

Para el desarrollo del proyecto se hará uso del microcontrolador Arduino Uno junto al escudo EVShield, el cuál, permite la conexión entre la placa y los sensores y actuadores LEGO. Debido a que Arduino no permite el uso de aplicaciones concurrentes se usará el escudo, previamente nombrado, junto al sistema operativo M2O2. Se trata de un sistema operativo en tiempo real que permite ejecutar aplicaciones multitarea escritas utilizando el lenguaje Ada.

El escudo EVShield dispone de una librería donde quedan definidas las funcionalidades de la plataforma. Para conseguir su correcto funcionamiento sobre el sistema operativo M2OS se ha realizado una adaptación de esta creando conectores (bindings).

Como comprobación final de la eficacia de la plataforma, se han desarrollado una serie de demostradores con el objetivo de obtener casos prácticos, y así demostrar el potencial de la plataforma.

Palabras clave.

Arduino, EVShield, LEGO Mindstorm, M2OS, Ada, Robótica.

Abstract.

The develop of technologies has allow the globalization of robotics in day to day of humans. Nowadays, robots are highly use both on a personal and industrial way, and lately, also with a great value in education.

The goal of this project is to build a framework to develop Ada apps with LEGO sensors and actuators in Arduino. This framework will be focus to be use by students and teachers developing concurrents apps for the Informatics Engineering degree of the University of Cantabria.

LEGO, in collaboration with ROBOTIX, has develop a robotics line named MindStorm in which we can found sensors and actuators that allow users to do lots of different projects. Also, this serie, focus the ease of use and a cheap price that everyone can afford.

To develop the project, we will use the Arduino Uno board with EVShield shield, which is needed to connect the LEGO sensors and actuators to the micro controller. Also, due to the fact that Arduino does not allow the develop of real time apps, we will use the multitask operating system M2OS.

M2OS is a real time operating system that allow the execution of multitask apps in Ada language.

EVShield shield provides a library where the functionality of the controller is defined. To achieve the goal of executing Ada apps with the LEGO sensors, I had generated bindings.

To finish with, some demos has been develop. Those demos shows the potential that this platform has. Also, they can help like as a guidebook for students and teachers for their projects.

Keywords.

Arduino, EVShield, LEGO Mindstorm, M2OS, Ada.

Índice

Índice de figuras

1. Introducción y objetivos

1.1. Motivación.

Desde hace años se ha podido observar como la importancia de la robótica en la vida de los humanos no ha hecho mas que aumentar; mientras que originalmente se centraba en un ámbito mas puramente focalizado en la automatización a un nivel industrial, en las últimas décadas hemos vivido una convergencia de esta disciplina con el bienestar de los humanos. El avance de diferentes ramas de la robótica como puede ser la domótica ha permitido este gran avance.

Las ventajas de los robots frente a las personas son claras, un robot será siempre más eficiente y mas preciso que cualquier ser humano realizando la tarea para la que haya sido diseñado (uso médico, militar, transporte, doméstico, etc.).

Es por esta razón que el estudio de esta disciplina ha estado creciendo paulatinamente. LEGO, en colaboración con ROBOTIX, ha desarrollado una linea de robótica denominada LEGO Mindstorm que tiene como objetivo de acercar la robótica al entorno educativo. Esta serie dispone de un amplio número de sensores y actuadores que pueden ser conectados y manejados a través de diferentes tecnologías de comunicación (I2C, UART, SPI, etc.). Asimismo, LEGO facilita el desarrollo de prototipos de sistemas robóticos (p.ej., cintas transportadoras, brazos robóticos o robots segway) mediante la construcción basada en bloques o ladrillos.

Por otro lado, las placas Arduino proporcionan una solución modular de bajo coste para el desarrollo de aplicaciones empotradas. Desde una perspectiva docente, resulta especialmente interesante la utilización de Arduino en prototipos construidos con piezas de LEGO ya que otorga un alto grado de libertad en el desarrollo, tanto a nivel software como hardware.

1.2. Objetivos.

Este trabajo tiene como principal objetivo conseguir la integración de sensores y actuadores Lego con placas Arduino. Para ello, existe un adaptador denominado EVShield que permite la conexión física entre la placa y los sensores/actuadores. Este adaptador dispone de una librería básica en C++ y utiliza el protocolo I2C para las comunicaciones. El proyecto busca desarrollar un marco que facilite el uso de la librería en aplicaciones escritas en lenguaje Ada. Para ello, se llevará a cabo la generación de conectores que permitan utilizar la librería C++ en este lenguaje. Dado que el software oficial de Arduino no soporta aplicaciones concurrentes, se pretende además que el software desarrollado se integre en un sistema operativo multitarea denominado M2OS.

Se busca crear una plataforma de fácil uso e intuitiva, la cuál pueda ser utilizada por futuros alumnos de la Universidad de Cantabria para la realización de diversas prácticas sobre aplicaciones concurrentes.

Otro de los objetivos ha sido el desarrollo de demostradores que puedan servir como guía para profesores y alumnos a la hora de plantear y desarrollar sus prácticas.

2. Herramientas, tecnologías y métodos

2.1. Tecnologías y lenguajes.

2.1.1. Arduino

Arduino dio comienzo cuando una serie de estudiantes del Instituto de Diseño Interactivo de Ivrea (Italia) quisieron dar una alternativa a los microcontroladores BASIC Stamp para que los alumnos con menor capacidad adquisitiva pudieran permitirse una plataforma de bajo coste ya que los BASIC Stamp costaban mas de 100 dolares.

Este proyecto dio vida a Arduino, lo que hoy en día es una de las mayores plataformas electrónicas de código abierto que se pueden encontrar. Una de las características que justifica la popularidad de Arduino es la gran flexibilidad que da al usuario para llevar a cabo una gran diversidad de tareas. [28]

Durante estos años, Arduino ha sido el centro de una gran cantidad de proyectos, los cuales van desde objetos para el día a día para estudiantes y aficionados hasta proyectos mas elaborados para el uso científico e industrial. Esto hace que exista una gran comunidad detrás de la plataforma. Podemos encontrar una gran variedad de personas que la dan uso, estudiantes, docentes, científicos, artistas, programadores...

En la actualidad, existen diversas alternativas a Arduino como pueden ser Raspberry Pi, Micro:bit, Parallax Basic Stamp, Netmedia's BX-24 o Phidgets entre otras, pero ninguna de ellas aportan las ventajas que aporta Arduino.[8]

- La relación calidad-precio que nos da esta plataforma es bastante satisfactoria, por menos de 50€ podrás encontrar una gran variedad de placas con diversas características.
- En relación con lo anterior, a diferencia de por ejemplo Raspberry que cuenta con un catalogo limitado, Arduino dispone de un gran número de placas con distintas características con las cuales podrás encontrar la adecuada para satisfacer tus objetivos.
- Mientras que muchos microcontroladores solo se encuentran para plataformas Windows, Arduino tiene un software multi-plataforma capaz de ser ejecutado tanto en Windows como en OSX y Linux.
- El IDE de Arduino proporciona una interfaz sencilla con la que los nuevos programadores se sentirán cómodos, al igual que aportará flexibilidad suficiente para el trabajo de los programadores mas expertos.
- Gracias a la extensa comunidad existente detrás de la plataforma, podremos encontrar una gran cantidad de librerías creadas por usuarios con multitud de funcionalidades ya programadas y que nos pueden ser de gran valor.
- Las placas Arduino son extensibles con unos componentes llamados Shields. Estos escudos aportan más funcionalidades a los dispositivos como pueden ser la conexión de pantallas táctiles, conectividad con radios, GPS o la posibilidad de conexión de sensores y actuadores LEGO por protocolo I2C.

2.1.2. Arduino Uno

Arduino Uno (Figura 1) es un placa basada en el microcontrolador ATmega328P. Además, es una de las placas mas famosas de la familia Arduino. Esta fama la ha conseguido debido a las siguientes características:

- Bajo coste. La placa se encuentra en la franja de los 20€, precio muy asequible para la inicialización de la gente en esta plataforma.
- Kits de desarrollo. Existen numerosos kits centrados en esta placa en los que se obtienen numerosos periféricos con los que realizar infinidad de proyectos.
- Compatibilidad. Sus características la hacen compatible con la gran mayoría de shields y accesorios de Arduino.
- Facilidad de uso. Como ya he comentado previamente, la comunidad es una de las grandes ventajas de Arduino y para esta placa no va a ser diferente ya que se encuentran gran cantidad de librerías o programas con los que experimentar. Por otra parte, incluso para los más novatos, implementar tus prototipos no requerirá de mucho esfuerzo.

Ahora, pasando a un aspecto más técnico, la placa Arduino Uno dispone de 14 pins digitales de los cuales, 6 pueden ser utilizados como salidas PWM (Modulación por ancho de pulsos), 6 entradas analógicas, un oscilador, una entrada USB, un conector de alimentación, una entrada ICSP (In Chip Serial Programmer) que da acceso a la memoria de programa y un botón reset.

La memoria Flash que dispone la placa es de 32 kb, de los cuales, 0.5 son utilizados por el bootloader (también se disponen de 2 KB de SRAM y 1 KB de EEPROM), de mientras, la frecuencia del reloj es de 16 Hz.

La placa funciona con una alimentación externa de entre 6 a 20 voltios (el rango recomendado por el fabricante es entre 7 y 12 V). Esta alimentación podrá ser obtenida a través del conector USB o con una fuente de alimentación externa. En la placa se dispone de 4 pines orientados al uso de la potencia:

- VIN. Este pin será la entrada de potencia cuando se utilice alimentación externa.
- 5V. Este pin proporciona una tensión de 5V los cuales serán regulados por la misma tarjeta.
- 3V3. Emite 3.3V. La alimentación de tensión a través tanto de este pin como del anterior no son del todo aconsejables ya que puentean el regulador y puede dañar la placa.
- GND. Pin de tierra.

Por último, la placa Arduino Uno dispone de una buena variedad de formas de comunicación, el microcontrolador ATmega328 proporciona una comunicación serie UART TTL (5V), que está disponible en los pines digitales 0 (RX) y 1 (TX). Por otra parte, el software de Arduino dispone de un monitor serie con el cual se puede enviar datos textuales desde y hacia la placa.[11]

Arduino está compuesto por un sistema de librerías las cuales aportan las funcionalidades para que los programadores desarrollen sus programas. En Arduino hay tres tipos de librerías:

- Core. La librería base o core forma parte del entorno de desarrollo de Arduino. El objetivo de esta es ocultar la complejidad de trabajar con un microprocesador. En ella se encuentran las funciones mas comunes como puede ser la escritura en los pines o la lectura de ellos.
- Estándar. Las librerías son las que no se incluyen como opción por defecto de nuestros sketches pero si que vienen incluidas en el IDE. Para cargar estas librerías en nuestro programa basta con declarar el *#include* al comienzo.

  La librería estándar SoftwareSerial habilita la comunicación serie en cualquiera de los pines digitales. Además, el ATmega328 soporta la comunicación I2C (TWI) y SPI. Se incluyen librerías para simplificar el uso de estos protocolos. En el caso de I2C se utilizará la librería estándar Wire y para SPI, la librería SPI.

- Contribuciones. Estas librerías son desarrolladas por los usuarios de Arduino y no son incluidas por defecto en el IDE sino que deben ser descargadas e instaladas. Un ejemplo de este tipo de librería es EVShield (Sección 3.1).

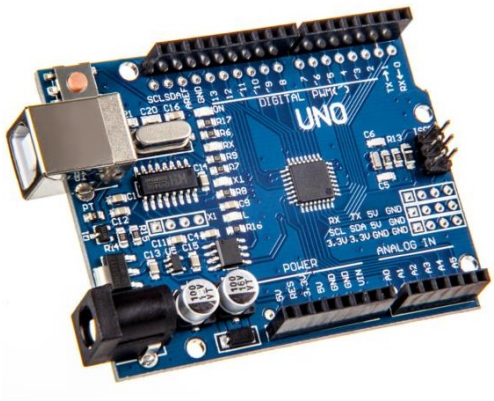

Figura 1: Microcontrolador Arduino Uno

2.1.3. IDE Arduino

Un IDE es un conjunto de herramientas software que facilitan al programador la tarea integral de desarrollo de software. El IDE de Arduino permite escribir, depurar, editar y grabar nuestros programas o "*sketches*" como son llamados en el mundo Arduino.

Una funcionalidad extra que nos da este IDE es la posibilidad de verificar el código, con lo que se comprobará que no existe ningún error sintáctico y en caso de ser correcto, compilar el código.

El IDE de Arduino dispone de un monitor (monitor serial) el cuál permite desde nuestro computador enviar y recibir datos textuales de la placa Arduino utilizando el cable USB.[7]

2.1.4. EVShield

EVShield[19] (Figura 2) es un escudo diseñado para Arduino con el objetivo de permitir controlar y operar con sensores NXT y EV3. Este shield ha sido desarrollado por la empresa MindSensors y es válido para Arduino Duemilanove, Arduino Uno o diversos clones de Arduino con las mismas características que los anteriores.

EVShield esta diseñado para ser conectada completamente sobre los pines de la placa Arduino, además, tiene sus propios pins con los que permite la conexión de otros shields sobre ella. EVShield utiliza el protocolo I2C como único modo de comunicación.

Para la alimentación del escudo, se dispone de dos opciones, a través de los pins de alimentación de Arduino obteniendo la energía del conector USB del microcontrolador o mediante una alimentación externa conectada a los pins de alimentación del escudo con la cual también se aporta energía a la placa Arduino Uno. Un detalle importante es que nos debemos asegurar de no exceder los 10.5 V de alimentación, puesto que un mayor voltaje puede dañar el escudo. Por otra parte, si se tienen conectados motores al escudo, se debe proporcionar un mínimo de 6.6 V.

Mindsensors dispone de un módulo de batería con el cual conectar nuestro EVShield, para el que se recomienda utilizar 6 pilas del tipo 6AA.

Por último, mirando desde un punto mas arquitectural, el escudo EVShield dispone de dos bancos (Bank-A y Bank-B). Cada banco tendrá su propia dirección I2C y dispondrán de dos entradas para motores o servos y otras dos para sensores. Además, EVShield dispone de 4 switchs; el switch reset está conectado directamente al botón reset de Arduino y será utilizado para reiniciar los programas, el switch GO suele ser utilizado para dar inicio a los programas y los switchs LEFT y RIGHT pueden ser utilizados a conveniencia del programador.[17]

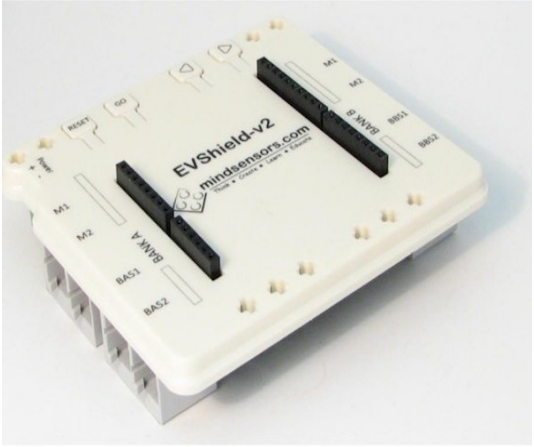

Figura 2: Escudo EVShield

2.1.5. I2C

I2C es un protocolo síncrono que usa solo dos cables, uno para el reloj (SCL) y otro para los datos (SDA). Esto mismo hace que el canal de datos sea utilizado tanto por el maestro como el esclavo. En maestro será el encargado de controlar el canal creando una señal de reloj.

El nombre I2C proviene de Inter-Integrated Circuit y fue desarrollado por Philips desplegando su versión 1.0 en 1992. Este bus es muy utilizado, especialmente en la comunicación entre microcontrolador y periféricos.[9]

2.1.6. M2OS

M2OS es un pequeño sistema operativo de tiempo real que permite ejecutar aplicaciones multitarea escritas utilizando el lenguaje Ada sobre microcontroladores con prestaciones muy limitadas (como es el caso de Arduino Uno).

Este sistema operativo implementa una planificación no expulsora que permite un alto aprovechamiento de la memoria. Esta política permite que el mismo área de stack pueda ser compartido por todas las tareas del sistema.

M2OS está escrito en Ada y se integra en el IDE GNAT Programming Studio (gnatstudio).

Actualmente, las placas soportadas por este sistema operativo son:

- Arduino Uno
- Epiphany
- STM32F4

M2OS ha sido desarrollado por el departamento de Ingeniería Informática y Electrónica de la Universidad de Cantabria.[10][4]

2.1.7. Ada

Ada es un lenguaje de programación estático y fuertemente tipado. Fue diseñado por un equipo dirigido por Jean Ichbiah de CII Honeywell Bull tras un encargo del Departamento de Defensa de los Estados Unidos buscando prescindir de los cientos de lenguajes que utilizaban, cada uno especializado para un proyecto y conseguir un único lenguaje capaz de realizar o incluso mejorar las tareas de todos ellos.

Ada es un lenguaje concurrente, multipropósito y orientado a objetos y con unos valores fundamentales que se apoyan en la eficiencia, seguridad, mantenibilidad y facilidad de uso. Hoy en día, Ada no es solo utilizado para la gran mayoría de hardware militar, sino que su uso es extenso también en la aeronáutica, industria aeroespacial y todo sistema crítico en el que un fallo del software pueda acarrear graves consecuencias para personas y/o infraestructuras.

Como ya se ha mencionado, Ada, fue diseñado con la seguridad en mente, con un objetivo de reducir cualquier tipo de error humano difícil de detectar y para ello hace uso de un tipado muy fuerte y chequeos en tiempo de compilación y ejecución.

El compilador utilizado para Ada es GNAT, este fue desarrollado por la Universidad de Nueva York y también patrocinado por el Departamento de Defensa. Esta basado en GCC y es software libre. Actualmente está mantenido por la empresa AdaCore.[1]

2.2. Herramientas.

2.2.1. Utilidades binarias AVR

AVR es un conjunto de herramientas y librerías utilizadas para crear aplicaciones para microcontroladores AVR. Esta colección dispone de compiladores, ensambladores, linkers y un conjunto de librerías matemáticas y de distintos lenguajes.

Estas herramientas han sido desarrolladas en su mayoría por GNU y por Microchip.[18]

2.2.2. GNAT Programming Studio

GNAT Programming Studio es un potente IDE que da soporte tanto para la creación de código y testeo, como para debugging y análisis de código. Además, permite automatizar la carga de aplicaciones en el microcontrolador (Arduino Uno)

GNAT Studio es un IDE multilenguaje que da soporte a Ada, SPARK, C, C++ y Python. Fue diseñado con la idea de dar la versatilidad necesaria al usuario consiguiendo poder adaptarse a todo tipo de proyecto de cualquier complejidad.

Este IDE es altamente extensible a través de scripting. Permite personalizar el entorno para los requerimientos de los usuarios de una forma sencilla. Así como también ofrece herramientas que facilitan en desarrollo cruzado de aplicaciones, la cuál es una técnica con la que se desarrolla la aplicación en un computador y se ejecuta en otro habitualmente menos potente.[2]

3. Librería EVShield.

Mindsensor, desarrollador del escudo EVShield proporciona una librería que permite utilizar los sensores y motores Lego conectados al escudo. En los siguientes apartados esta será explicada.

3.1. Introducción a la librería.

EVShield utiliza el IDE de Arduino como interfaz de programación, en caso de no tener previamente este IDE instalado se debe proceder a su instalación. El Anexo A desarrolla el proceso que se debe llevar a cabo para este objetivo, de mientras, en el Anexo B se puede ver el proceso de instalación para la librería EVShield.

Una vez seguidos los pasos de instalación se puede proceder a la ejecución de un programa de prueba para comprobar el correcto funcionamiento tanto del IDE como de la librería. Para ello se debe abrir el IDE de Arduino y buscar en el menú la ruta $File \rightarrow Examples \rightarrow EVShield$, tras esto se obtiene un desplegable donde aparecen 3 opciones; EVShield_examples, EVShield_test y WifiInterfaceForPitStorms_examples. En los dos primeros se encuentran diversos programas de ejemplo con los que se puede comprobar el correcto funcionamiento del escudo. Uno de los ejemplos será bastante adecuado para este propósito, el programa llamado print_bank_info dentro de la carpeta test. Este nos sacará la información básica de los dos bancos que posee el escudo EVShield.

Para ejecutar el programa, teniendo la placa Arduino Uno con el EVShield conectada al computador, simplemente se debe hacer el cargado del programa de la forma que se observa en la Figura 3.

Figura 3: Botón para cargar un programa en el IDE de Arduino

Como se puede observar en este menú que se encuentra en la esquina superior izquierda del IDE de Arduino, el segundo botón nos permitirá cargar el programa en el microcontrolador. El primer botón verifica que nuestro código no tenga ningún error sintáctico mientras que los tres últimos permiten crear un nuevo proyecto, abrir uno ya existente o guardar los cambios realizados en el proyecto actual respectivamente.

Tras esto se debe abrir el serial monitor que es por donde se reciben las salidas de nuestro programa. Para ello se puede acceder a través del menu $Tools \rightarrow Serial\ monitor$ o con la combinación de teclas Ctrl+Shift+M . Una vez ya le tenemos abierto es importante configurar la tasa de baudios correcta, el baudaje es el número de unidades de señal por segundo, es decir, la velocidad a la que se transmiten los datos. Para elegir la correcta se puede encontrar en la parte inferior derecha del serial monitor un desplegable en el que se deberá escoger una tasa de 115200 baud.

En la Figura 4 se observa como el programa nos indica que se va a esperar a que se pulse el switch GO del escudo EVShield para continuar y una vez presionado, se empezará a imprimir información tanto de los dos bancos como del voltaje que esta obteniendo de la alimentación.

Ya con esto habremos conseguido ejecutar nuestro primer programa sobre EVShield. La librería nos proporciona varios ejemplos mas con los que poder probar diversas funcionalidades como puede ser la aplicación blinkers, con la cuál, utilizando los switchs Right y Left se encenderá el led del banco respectivo al switch pulsado y también podremos empezar a probar los diversos sensores y motores que admite con unos programas de prueba bastante sencillos.

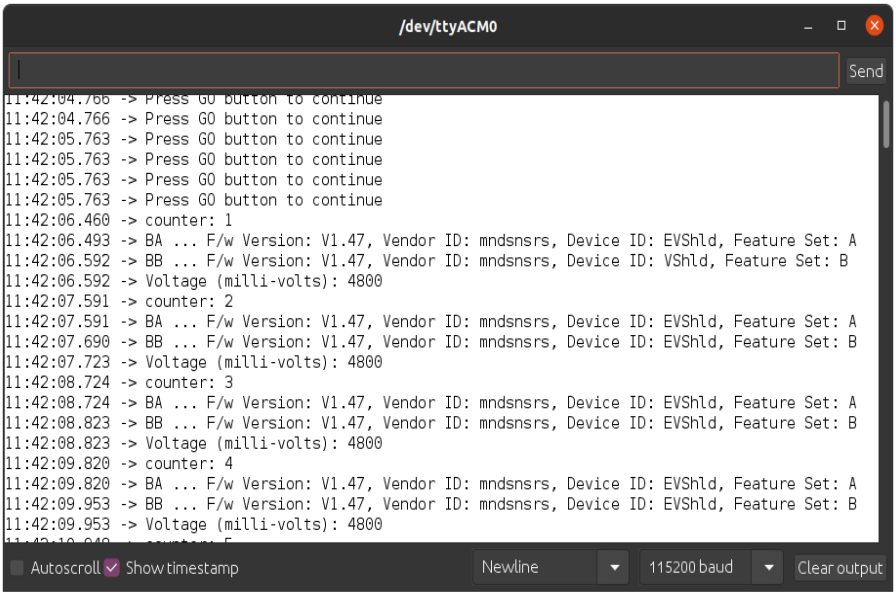

Figura 4: Serial monitor

Una vez comprobado el correcto funcionamiento de la librería vamos a pasar a comentar como está estructurada. La librería[17] se puede dividir en 3 partes:

1. Ficheros de cabeceras y clases. Sobre estos ficheros recae la funcionalidad del escudo y todos los componentes/sensores que se pueden conectar.
2. Programas de prueba. Sencillos programas para comprobar el correcto funcionamiento de la librería o para utilizarlos como referencia para proyectos propios.
3. Documentación. La librería proporciona una sencilla interfaz web con referencias sobre su estructura de clases.[16]

La librería EVShield soporta un gran número de sensores y actuadores de las distintas versiones de los LEGO Mindstorm. En este proyecto se utilizarán varios de los más populares, los cuales son:

- EV3Touch.
- EV3Ultrasonic.
- EV3Color.
- NXTColor.
- Motores EV3 y NXT.

Por esto mismo, me limitaré a hablar sobre los ficheros y clases utilizados por estos.

3.2. Estructura de la librería.

Enfocándonos en la jerarquía de clases, encontramos dos clases principales de las cuales heredan las demás. Estas dos clases son SoftI2CMaster y BaseI2CDevice. Estas clases implementan la interfaz software o hardware del protocolo I2C.

A pesar que los programas funcionarán de ambas formas (Hardware o software), el fabricante nos recomienda utilizar siempre la versión Hardware del protocolo. La diferencia entre ambos sería que mientras

en la versión Hardware, se utilizarán registros específicos para la comunicación de los datos, con la versión Software no se utilizarán esos registros por lo que se necesitaran realizar una serie de operaciones con las que harán la comunicación menos eficiente.

Dentro de la clase BaseI2CDevice se encuentra un constructor al cual se le pasará la dirección del dispositivo. Las direcciones utilizadas por los bancos del escudo EVShield por defecto son el 0x34 para el banco A y 0x36 para el banco B.

```
BaseI2CDevice::BaseI2CDevice (uint8_t i2c_address)
```

Así, esta clase también tiene otros métodos que son capaces de validar si un dispositivo esta conectado a un bus I2C o de retornar la dirección de memoria de la instancia I2C.

```
bool BaseI2CDevice::checkAddress ( )
uint8_t BaseI2CDevice::getAddress ( )
```

Por último, en esta clase también están definidos una serie de métodos con los que se puede conseguir diversa información de los bancos del escudo como puede ser *getFirmwareVersion()* que retorna la versión del firmware actual o *getVendorID()* que devuelve el ID del desarrollador del dispositivo.

Por debajo de las clases BaseI2CDevice y SoftI2CMaster, se encuentra la clase EVShieldI2C (Figura 5), esta clase será la encargada de implementar la interfaz I2C con la que conseguir la comunicación entre el escudo y la placa Arduino.

Esta interfaz es creada heredando los métodos definidos en la interfaz Software o Hardware elegida. Contiene un constructor (*EVShieldI2C()*).

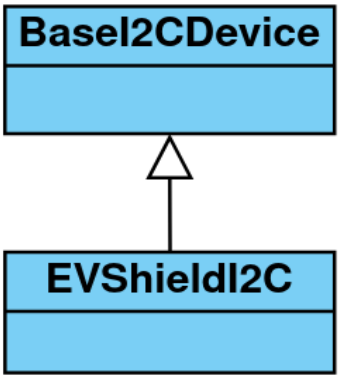

Figura 5: Diagrama de herencia para la clase EVShieldI2C

Ahora en el siguiente nivel se encuentra la clase EVShieldBank (Figura 6), esta implementa los métodos utilizados por los bancos del escudo EVShield. Esta clase hereda los funciones de las clases anteriormente comentadas y a su vez implementa diversos métodos con las cuales se controla los bancos del escudo.

Cada banco del escudo posee dos entradas para motores o servos, dos entradas para sensores y un led. Dentro de la clase EVShieldBank se encuentra tanto el método utilizado para controlar el led como todas las funciones relacionadas con el uso de los motores.

Con el método *bool ledSetRGB (uint8_t R, uint8_t G, uint8_t B)* se podrá encender el led que posee el banco. Este led podrá ser encendido con cualquier color del espectro RGB, para ello, se deberá pasar los parámetros adecuados para dejarlo a interés del programador.

Por otra parte, en esta clase se encuentran las funciones necesarias para el uso de los motores. Entre estos métodos se pueden destacar:

- *void motorRunUnlimited (SH_Motor which_motors, SH_Direction direction, int speed)*. Esta función recibe 3 parámetros, el primero se refiere a cual de los dos motores es el que recibe la acción, el segundo a que acción debe realizar (marcha adelante, marcha atrás) y por último, cuál es la velocidad a la cuál la debe ejecutar. Esta acción se ejecutará hasta recibir otra orden que la modifique.
- *uint8_t motorRunRotations (SH_Motor which_motors, SH_Direction direction, int speed, long rotations, SH_Completion_Wait wait_for_completion, SH_Next_Action next_action)*. Esta función es similar a la interior pero con la diferencia de que además de lo descrito para la función anterior, esta, recibe el número de rotaciones que debe realizar el motor, como debe actuar al finalizarlas y si debe esperar a completar esas funciones para seguir con el programa o si este puede seguir siendo ejecutado mientras el motor está en uso.
- *bool motorStop (SH_Motor which_motors, SH_Next_Action next_action)*. Esta función parará el motor indicado por parámetro.

El banco B posee diferencias arquitecturales con el banco A, debido a ello, la clase EVShieldBankB (Figura 6) redefine alguno de los métodos de la clase EVShieldBank.

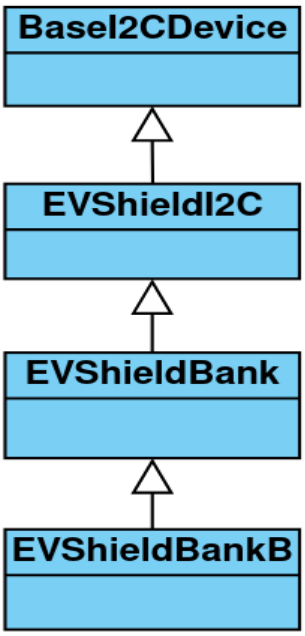

Figura 6: Diagrama de herencia para la clases EVShieldBank y EVShieldBankB

EVShield (Figura 7) es la clase encargada de definir las funcionalidades del escudo. Esta clase tiene 3 atributos, uno que indica el protocolo I2C que se va a utilizar y dos objetos, uno de la clase EVShieldBank y otro de EVShieldBankB. Estos atributos serán inicializados a través de las dos siguientes funciones:

- EVShield es el constructor de la clase EVShield, será e encargado de inicializar los bancos y para ello se le pasarán las direcciones I2C de ambos en caso de querer unas direcciones personalizadas. Si dejas vacío el constructor se utilizan las direcciones por defecto (0x34 para el Bank-A y 0x36 para el Bank-B).

```
EVShield::EVShield(uint8_t i2c_address_a = SH_Bank_A,
                   uint8_t i2c_address_b = SH_Bank_B);
```

- init es la función encargada de inicializar el protocolo I2C y los timers. La función, en caso de no recibir ningún parámetro, toma por defecto el protocolo Hardware, que como ya he comentado antes es el recomendado.

```
void EVShield::init(SH_Protocols protocol = SH_HardwareI2C)
```

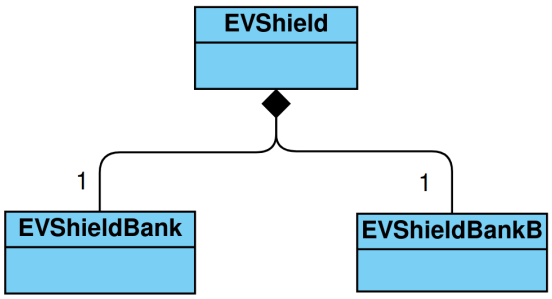

Figura 7: Clase EVShield

Con estas clases tenemos las funcionalidades necesarias para el correcto funcionamiento de el escudo, ahora pasaré a comentar las clases utilizadas para los sensores.

EVs_NXTColor es la clase encargada del uso del sensor NXTColor.

El constructor *EVs_NXTColor::EVs_NXTColor (EVShield* shield, SH_BankPort bp)* será el encargado de inicializar el sensor. Los parámetros que debe recibir son un puntero al escudo donde es utilizado el sensor y el identificador del puerto al que ha sido conectado.

Además contiene una función init() con la que inicializar el sensor y las funciones setType(), readValue() y readColor() las cuales marcaran el uso del sensor.

La clase EVshieldUART define el uso de los sensores de tipo UART. Estos son entre otros, los sensores EV3Color, EV3Touch y EV3Ultrasonic, con sus respectivas clases EVs_EV3Color, EVs_EV3Touch y EVs_EV3Ultrasonic. Los sensores de tipo UART extienden a la clase EVShieldUART, como se muestra en la Figura 8

Dentro de la clase EVShieldUART se encuentra el constructor que permite inicializar el dispositivo.

```
EVShieldUART::EVShieldUART (EVShield* shield,
            SH_BankPort bp)
```

Al igual que el comentado anteriormente de la clase EVs_NXTColor, el constructor recibe como parámetros el banco y el identificador del puerto en el que se encuentra conectado el sensor. Esta clase además define métodos con los que poder leer valores de los dispositivos con funciones como *readLocationByte()*.

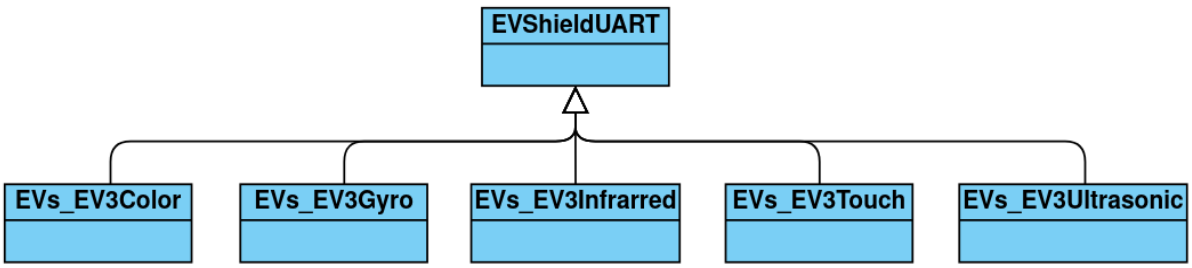

Figura 8: Diagrama de herencia para las clases EVs_EV3Color, EVs_EV3Touch y EVs_EV3Ultrasonic

Las clases EVs_EV3Color, EVs_EV3Touch y EVs_EV3Ultrasonic definen los métodos necesarios para el correcto funcionamiento de los sensores. En ellas se pueden encontrar funciones como *bool isPressed()* dentro de EVs_EV3Touch y con la cuál se comprueba si el sensor de contacto está presionado o *float getDist()* definida en la clase EVs_EV3Ultrasonic y con la que se obtiene la distancia medida por el sensor de ultrasonidos.

4. Adaptación de la librería para la solución propuesta.

Como se comentó en la sección de objetivos, en este proyecto se ha buscado desarrollar un marco de desarrollo de aplicaciones Ada con sensores/actuadores Lego en Arduino, y debido al hecho de que Arduino no permite la ejecución de tareas concurrentes se ha integrado el trabajo sobre el sistema operativo multitarea M2OS.

Disponiendo de la librería desarrollada por Mindsensor y previamente comentada, se ha procedido a adaptarla para el correcto funcionamiento dentro de este sistema operativo.

Para ejemplificar la solución propuesta se ha realizado un diagrama de capas (mostrado en la Figura 9) con el cuál poder entender la arquitectura de una aplicación.

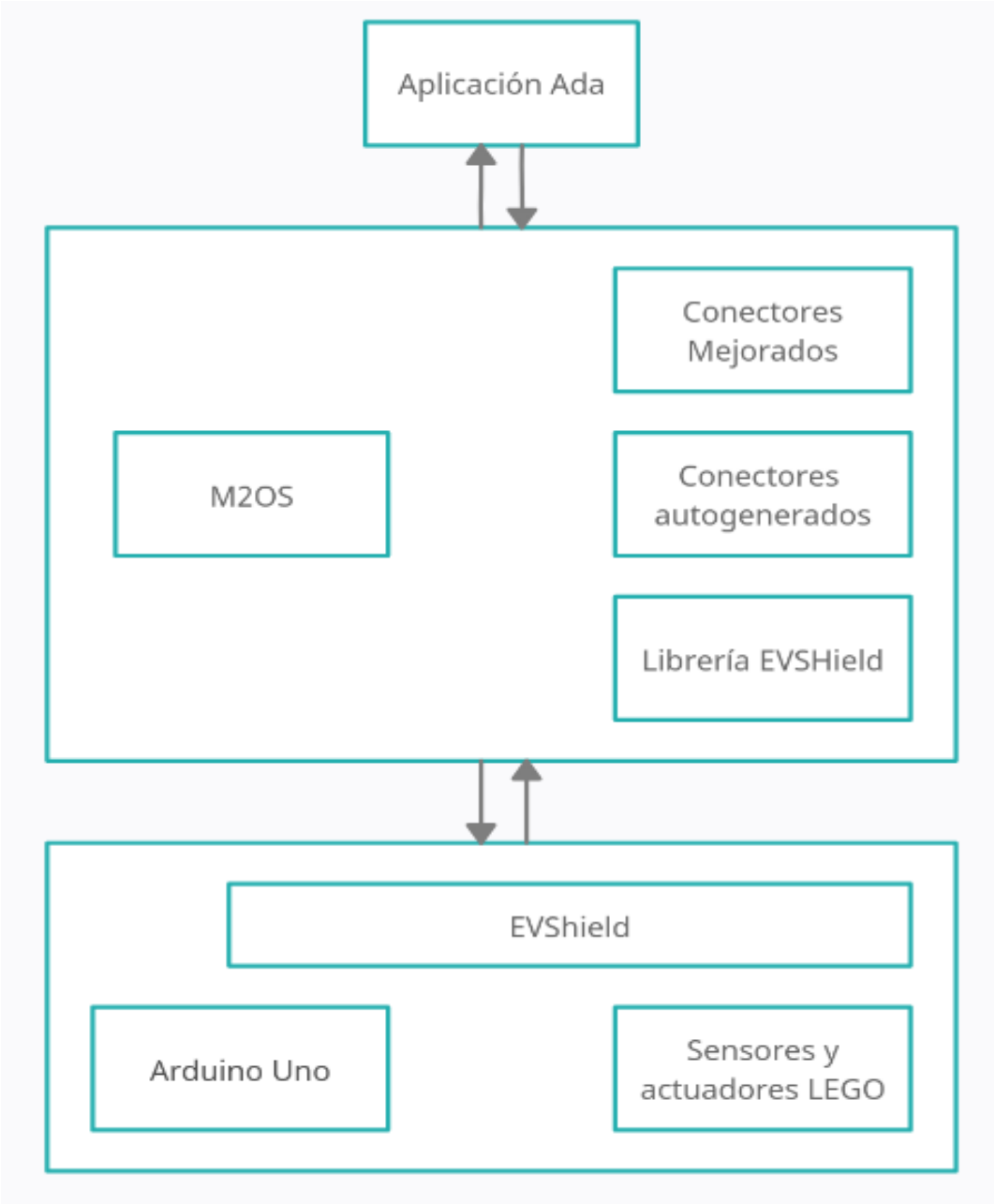

Figura 9: Diagrama de capas de la solución propuesta.

En lo alto del diagrama se encuentra la aplicación Ada, en ella se desarrollará el código necesario para llevar a cabo el caso de uso requerido. Esta aplicación se comunicará con el siguiente nivel, en el que por un lado encontramos el sistema operativo M2OS el cuál nos permite la ejecución de aplicaciones concurrentes y por otro lado los conectores que permiten a la aplicación Ada acceder a las funcionalidades de la librería EVShield.

En el último nivel del diagrama se encuentra los elementos Hardware necesarios para la ejecución de la aplicación. El escudo EVShield es el encargado de realizar la conexión entre el microcontrolador EVShield y los sensores y actuadores LEGO. Por otra parte, a través del sistema operativo M2OS, también podremos acceder al microcontrolador Arduino Uno sin necesidad de tener por medio el escudo EVShield.

En esta sección, se desarrolla la creación de los conectores (bindings). Gracias a los conectores somos capaces de invocar una función escrita en otro lenguaje.

4.1. Interconexión con C++

Ada dispone de una serie de directivas de compilador (pragmas) con las cuales poder invocar funciones desarrolladas en otros lenguajes. Entre ellas se encuentra *pragma import*, esta directiva da la habilidad a este lenguaje a realizar llamadas al lenguaje que se quiera.

Recordando que la librería original de EVShield esta desarrollada en C++, esta herramienta nos dará la posibilidad de dar una capa de abstracción con la que conseguir la conexión entre estos dos lenguajes.[13]

Por otra parte, además de estas directivas, Ada aporta un conjunto de librerías con las que facilitar la interconexión con algunos lenguajes. Estas librerías declaran una serie de tipos de dato o entidades propios de cada lenguaje. Se encuentran estas librerías para los lenguajes C/C++, Fortran y COBOL.[14]

Centrándonos en la que nos hará falta que no es otra que la de c/c++, se puede observar que tiene este aspecto:

```
package Interfaces.C is
    pragma No_Elaboration_Code_All;
    pragma Pure;

    type int   is new Integer;
    type short is new Short_Integer;
    type long  is new Long_Integer;

    type unsigned       is mod 2 ** int'Size;
    type unsigned_short is mod 2 ** short'Size;
    type unsigned_long  is mod 2 ** long'Size;

    ....

end Interfaces.C;
```

Listado 1: Interfaz Ada para tipos de datos C/C++.

Se puede observar como en el Listado 1 se definen tipos de datos Ada que son equivalentes a los de los lenguajes C/C++.

4.2. Generación automática de conectores con C++

Para la realización de la conexión entre Ada y la librería EVShield desarrollada en C++ existen dos posibilidades. La generación manual de los pragma con los que conseguir poder llamar a las funciones necesarias o la generación automática de estos ejecutando un generador automático.

Para la realización de esta tarea será utilizada la versión automática ya que aportará una reducción de la complejidad y sobre todo un ahorro de tiempo, aunque como mas tarde se verá, esta herramienta puede inducir a errores en el código.

En Sección 2.2.1 se ha hablado de las utilidades binarias AVR, un conjunto de herramientas para creación de aplicaciones en microcontroladores. Entre estas herramientas se encuentra incluida el compilador *avr-g++*. Invocando la opción de compilación *-fdump-ada-spec* y especificando el fichero de cabeceras que queramos realizar la conexión conseguiremos los ficheros .ads con los cuales conseguimos la conexión con C++ y los cuales deberemos añadir al sistema operativo M2OS.

Esta herramienta también es capaz de resolver las dependencias con otras clases que tengan los ficheros de los cuales se quieren realizar las conexiones.

Así, el Listado 2 muestra el comando completo para conseguir la generación automática de los conectores C++ será el siguiente:

```
/home/daniel/arduino-1.8.13/hardware/tools/avr/bin/avr-gcc -c -g -Os -w -std=gnu++11
-fpermissive -fno-exceptions -ffunction-sections -fdata-sections
-fno-threadsafe-statics -Wno-error=narrowing -MMD -flto -mmcu=atmega328p
-DF_CPU=16000000L -DARDUINO=10813 -DARDUINO_AVR_UNO -DARDUINO_ARCH_AVR
-I/home/daniel/arduino-1.8.13/hardware/arduino/avr/cores/arduino
-I/home/daniel/arduino-1.8.13/hardware/arduino/avr/variants/standard
-I/home/daniel/arduino-1.8.13/hardware/arduino/avr/libraries/Wire/src
-I/home/daniel/Arduino/libraries/EVShield -fdump-ada-spec <fichero.h>
```

Listado 2: Comando utilizado para la generación automática de los conectores.

Donde el último parámetro fichero.h será el fichero de cabeceras del que se quiere hacer la conexión.

Como se puede observar en el Listado 2, es necesario incluir diversos parámetros y características de nuestra placa, además de localizaciones de librerías Arduino. Estos parámetros son necesarios con el objetivo de conseguir las correctas macros/funcionalidades de la placa en la librería EVShield.

Para conseguir este comando de una forma sencilla se recomienda ver como el IDE de Arduino compila los proyectos para conseguir los binarios utilizados al cargar un programa al controlador. Para ello se debe entrar en el Ide de Arduino y en $File \rightarrow Preferences$ hay que marcar en la opción *Show verbose output during:* la casilla *Compilation*.

Tras guardar y probar a cargar un programa se consigue observar en la consola del IDE como se compilan los ficheros y de ahí se puede extraer un comando similar al superior donde se muestran las características de nuestra placa y ficheros Arduino que son necesarios en tiempo de compilación.

Este comando será ejecutado para los ficheros:

- EVShield.h
- EVShieldUART.h
- EVs_EV3Ultrasonic.h
- EVs_EV3Touch.h

- EVs_NXTColor.h
- EVs_EV3Color.h

4.3. Integración de la librería en M2OS

Como ya se ha comentado previamente en la Sección 2.1.6, M2OS es un sistema operativo ligero que permite ejecutar aplicaciones concurrentes desarrolladas en el lenguaje Ada, por lo tanto es el sistema operativo perfecto para desarrollar el objetivo del proyecto.

Una vez conseguido tener los ficheros necesarios para realizar la conexión entre Ada y la librería EVShield, se deben ver como integrarlos en este SO.

Bajo la ruta */arch/arduino_uno/drivers/libcore* dentro de la carpeta principal de M2OS, se encuentra una carpeta en la cuál se hallan las librerías con las cuales se han ido añadiendo las diversas funcionalidades Arduino de las que disponía el sistema operativo como son los conectores correspondientes a la librería estándar de Arduino y librerías para el manejo de distintos sensores, actuadores y kits robóticos, como puede ser Zowi, un robot inteligente de la marca BQ utilizado en el ámbito educacional.

Por otra parte, en este directorio también se puede encontrar un fichero Makefile, este es el encargado de generar la librería de Arduino *lib_core_arduino.a* con las funciones Arduino.

En este fichero Makefile se deben especificar las nuevas librerías que debe soportar M2OS (que son las mencionadas previamente: EVShield.h, EVShieldUART.h, EVs_EV3Ultrasonic.h, EVs_EV3Touch.h, EVs_NXTColor.h, EVs_EV3Color.h).

4.4. Errores en la generación de los conectores.

Aunque la generación automática de los conectores con C++ es bastante beneficiosa en relación a eficiencia de trabajo, la herramienta no es perfecta, puede inducir a errores en el código.

Tras la generación de los conectores y la integración de M2OS se pasó a comprobar el correcto funcionamiento de la librería, detectando que el banco B no funcionaba de la forma esperada. Este caso fue comprobado con el programa de prueba *EVShield_Print_Info*. Este debe imprimir información básica de cada uno de los bancos, fallando en la ejecución en el banco B.
Tras una búsqueda del error se concluyó con que fue un error en la creación del conector de la clase SoftI2CDevice, que como se ha comentado en el Sección 3.2, es la encargada de definir el protocolo Software I2C. En ella se encuentra la variable *initialized*, la cuál es un booleano. Al crear automáticamente los conectores, esta se creó con el tipo de dato *Extension.bool* el cuál es definido por Ada como una variable que ocupa 2 bytes, mientras que el tipo de dato booleano en C++ ocupa 1 byte. Esto hace que toda la estructura de memoria del banco B este 1 byte desplazada en memoria lo que causa los errores en el acceso al citado banco.

Para corregir este fallo se optó por utilizar el tipo *Boolean8* definido en el paquete Arduino.ads.

La estrategia utilizada para encontrar el error fue imprimir las direcciones de memoria en las que empezaba y acababan las estructuras de control de los bancos A y B tanto desde el IDE de Arduino como desde M2OS, lo que permitió descubrir el problema y, posteriormente, identificar el campo de la estructura que provocaba el desplazamiento.

4.5. Re-adaptación de los conectores

Este proyecto sigue un objetivo de elaborar un marco de desarrollo el cuál pueda ser utilizado por futuros alumnos del Grado de Informática de la Universidad de Cantabria para la elaboración de diversas

prácticas de programación concurrente, por lo tanto, no de los objetivos es la facilidad de uso del marco.

Otra de las desventajas de la generación automática de los conectores es que el código generado es poco natural. Por ello se ha llevado a cabo una readaptación de los conectores con la que se ha buscado mejorar la facilidad de uso y entendimiento de la librería. Para ello se ha usado una estrategia que implica el uso de paquetes genéricos.

Ada permite crear unidades genéricas, las cuales dan la posibilidad de crear instancias distintas de una misma unidad. Para entender mejor esto vamos a poner un caso práctico. Se dispone en un proyecto dos motores, lo que los hace diferentes es el banco y el puerto al que estén conectados. Un genérico nos da habilidad de tener dos instancias distintas, una para cada motor, de una misma unidad, el motor, y cada una de ellas diferenciadas por el puerto al que se conectan al EVShield. Así, un paquete genérico es aquel que permite la generalidad de los datos. El código correspondiente a un paquete genérico en Ada se escribe en dos ficheros: especificación (extensión .ads) e implementación (extensión .adb).[27]

La estrategia seguida ha sido similar en todos los paquetes. Para describirla se utiliza como ejemplo la adaptación del paquete correspondiente al sensor EV3Touch puesto que es uno de los más sencillos y permite apreciar el uso de los paquetes genéricos.

Conector generado con la herramienta automática:

```
1  package EVs_EV3Touch_h is
2     package Class_EVs_EV3Touch is
3        type EVs_EV3Touch is limited record
4           parent : aliased EVShieldUART_h.Class_EVShieldUART.EVShieldUART;
5        end record;
6        pragma Import (CPP, EVs_EV3Touch);
7
8        function New_EV3Touch return EVs_EV3Touch;
9        pragma CPP_Constructor (New_EV3Touch, "_ZN12EVShieldUARTC1Ev");
10
11       function init
12         (this : access EVs_EV3Touch;
13          shield : access EVShield_h.Class_EVShield.EVShield;
14          bp : SHDefines_h.SH_BankPort) return Extensions.bool;
15       pragma Import (CPP, init, "_ZN12EVs_EV3Touch4initEP8EVShield11SH_BankPort");
16
17       function isPressed (this : access EVs_EV3Touch) return Extensions.bool;
18       pragma Import (CPP, isPressed, "_ZN12EVs_EV3Touch9isPressedEv");
19
20       function getBumpCount (this : access EVs_EV3Touch) return int;
21       pragma Import (CPP, getBumpCount, "_ZN12EVs_EV3Touch12getBumpCountEv");
22
23       function resetBumpCount (this : access EVs_EV3Touch) return Extensions.bool;
24       pragma Import (CPP, resetBumpCount, "_ZN12EVs_EV3Touch14resetBumpCountEv");
25    end;
26    use Class_EVs_EV3Touch;
27 end EVs_EV3Touch_h;
```

Listado 3: Conector para el sensor EV3Touch generado con la herramienta automática.

El Listado 3 muestra el conector generado por la herramienta automática del compilador *avr-g++*. En este paquete encontramos un constructor vacío que genera un objeto del sensor y cuatro funciones, la primera de ellas, init() se encarga de inicializar el sensor. Esta recibe como parámetros su propio objeto,

un puntero al escudo EVShield y el puerto al que ha sido conectado. Después se encuentran tres funciones que realizan las funcionalidades del sensor, todas ellas deben recibir por parámetro el objeto del sensor de contacto donde se deben ejecutar.

El Listado 4 y el Listado 5 muestran los paquetes generados manualmente para la re-adaptación de los conectores.

Paquete de especificación:

```
1  with Interfaces.C; use Interfaces.C;
2  with EVShieldUART_h;
3  with SHDefines_h;
4  with Interfaces.C.Extensions;
5  with Arduino.EVShield;
6
7
8  package Arduino.EVShield.EV3Touch is
9
10    generic
11       Bp : SHDefines_h.SH_BankPort;
12    package Touch is
13
14       -- check if the touch sensor is pressed (or bumped)
15       function Is_Pressed return Extensions.bool;
16
17       -- You can get bump count for EV3Touch Sensor (an incremental
18       -- pressed value) this function will return the bump count since last reset.
19       -- (The max value of bumpCount is 254, after that it will not increment).
20       -- Programming Tip:
21       -- If you don't want to wait to see if button is pressed,
22       -- use this bump count,
23       -- store the old bumpCount in a variable and see if the new
24       -- bumpCount is larger than the old value.
25       function Get_Bump_Count return Interfaces.C.int;
26
27       -- reset the bump count and start the incremental bumps from zero
28       function Reset_Bump_Count return Extensions.bool;
29    end Touch;
30
31 end Arduino.EVShield.EV3Touch;
```

Listado 4: Paquete de especificación para la readaptación del conector del sensor EV3Touch.

Paquete de implementación:

```
1  pragma Ada_2005;
2  pragma Style_Checks (Off);
3
4  with Interfaces.C; use Interfaces.C;
5  with EVShieldUART_h;
6  with SHDefines_h;
7  with Interfaces.C.Extensions;
8  with EVs_EV3Touch_h;
9
10 package body Arduino.EVShield.EV3Touch is
11
12    package body Touch is
13       use type EVs_EV3Touch_h.Class_EVs_EV3Touch.EVs_EV3Touch;
14       Touch : aliased EVs_EV3Touch_h.Class_EVs_EV3Touch.EVs_EV3Touch :=
15             EVs_EV3Touch_h.Class_EVs_EV3Touch.New_EV3Touch;
16
17       Aux : Extensions.bool:= EVs_EV3Touch_h.Class_EVs_EV3Touch.init(Touch'Access,
18             Arduino.EVShield.EVShield_Obj'Access, Bp);
19
20       function Is_Pressed return Extensions.bool is
21       begin
22          return EVs_EV3Touch_h.Class_EVs_EV3Touch.isPressed(Touch'Access);
23       end Is_Pressed;
24
25       function Get_Bump_Count return int is
26       begin
27          return EVs_EV3Touch_h.Class_EVs_EV3Touch.getBumpCount(Touch'Access);
28       end Get_Bump_Count;
29
30       function Reset_Bump_Count return Extensions.bool is
31       begin
32          return EVs_EV3Touch_h.Class_EVs_EV3Touch.resetBumpCount(Touch'Access);
33       end Reset_Bump_Count;
34
35    end Touch;
36
37 end Arduino.EVShield.EV3Touch;
```

Listado 5: Paquete de implementación para la readaptación del conector del sensor EV3Touch.

El procedimiento realizado al ejecutar la re-adaptación (Listado 4 y Listado 5) ha sido añadir una capa de abstracción a los conectores creados automáticamente con la herramienta AVR con la que tener la posibilidad de conseguir varias instancias de una misma unidad con el uso de un paquete genérico.

El paquete Arduino.EVShield.EV3Touch define el paquete genérico que tiene como único parámetro el identificador del banco/puerto en el que se encuentra conectado el sensor. Diferentes instancias de este paquete con diferentes valores del parámetro genérico permite manejar diferentes sensores.

Esta implementación además evita al usuario tener que crear el objeto e inicializarle con los métodos anteriormente comentados. Esto se realiza de forma transparente al usuario durante la elaboración del paquete de implementación (Listado 5) en las líneas 14-15 y 17-18. Este paquete además, realiza las llamadas a las funciones pertinentes con el objeto del sensor, lo cuál hace ahorrarse este parámetro al

usuario lo que facilita el uso de la librería.

Para comprobar los cambios obtenidos al realizar la re-adaptación de los conectores, se describe el programa de prueba EVShield_Touch_Sensor. Primero se encuentra el programa que utiliza los conectores creados de forma automática.

```
1  with Interfaces.C;
2  with M2.Direct_IO;
3  with EVs_EV3Touch_h; use EVs_EV3Touch_h;
4  with EVShield_h; use EVShield_h;
5  with SHDefines_h; use SHDefines_h;
6  with Interfaces.C.Extensions;
7
8  procedure EVShield_Touch_Sensor is
9     package DIO renames M2.Direct_IO;
10
11    use type Arduino.Time_MS, SHDefines_h.SH_Protocols, Interfaces.C.Extensions.bool;
12    use type EVShield_h.Class_EVShield.EVShield,
13         EVs_EV3Touch_h.Class_EVs_EV3Touch.EVs_EV3Touch;
14
15    -- Create shield object
16    EVShield : aliased EVShield_h.Class_EVShield.EVShield :=
17         EVShield_h.Class_EVShield.New_EVShield (16#34#,16#36#);
18
19    -- Program variables
20    Touch : Interfaces.C.Extensions.bool;
21    Count : Interfaces.C.int;
22    Aux : Interfaces.C.Extensions.bool;
23
24    -- Create our sensor for use in this program
25    MyTouch : aliased EVs_EV3Touch_h.Class_EVs_EV3Touch.EVs_EV3Touch;
26 begin
27    -- initialize the shield i2c interface.
28    EVShield_h.Class_EVShield.init(EVShield'Access, SHDefines_h.SH_HardwareI2C);
29    -- initialize the EV3Touch sensor object
30    Aux := EVs_EV3Touch_h.Class_EVs_EV3Touch.init(MyTouch'Access,
31         EVShield'Access, SHDefines_h.SH_BBS1);
32    loop
33       -- get the reading(s) from sensor
34       Touch := EVs_EV3Touch_h.Class_EVs_EV3Touch.isPressed(myTouch'Access);
35       Count := EVs_EV3Touch_h.Class_EVs_EV3Touch.getBumpCount(myTouch'Access);
36       -- print the sensor value(s)
37       if Touch then
38          DIO.Put_Line ("Button is pressed!");
39       end if;
40       DIO.Put ("Count: "); DIO.Put_Line (Count'Img);
41       -- wait for some time
42       Arduino.C_Delay (1000);
43    end loop;
44 end EVShield_Touch_Sensor;
```

Listado 6: Programa EVShield_Touch_Sensor desarrollado con los conectores autogenerados.

Código del programa de prueba EVShield_Touch_Sensor tras la aplicación de los genéricos:

```
1  with Interfaces.C;
2
3  with Arduino.Wire;
4  with M2.Direct_IO;
5  with AdaX_Dispatching_Stack_Sharing;
6  with Arduino.EVShield.EV3Touch;
7  with Arduino.EVShield;
8  with Interfaces.C.Extensions;
9
10 procedure EvShield_Touch_Sensor is
11    package DIO renames M2.Direct_IO;
12
13    use type Arduino.Time_MS, Interfaces.C.Extensions.Bool;
14
15    -- Program variables
16    Touch : Interfaces.C.Extensions.Bool;
17    Count : Interfaces.C.Int;
18
19    package Touch_Sensor is new Arduino.EVShield.EV3Touch.Touch (Arduino.EVShield.SH_BAS1);
20
21 begin
22
23    loop
24       -- get the reading(s) from sensor
25       Touch := Touch_Sensor.Is_Pressed;
26       Count := Touch_Sensor.Get_Bump_Count;
27
28       -- print the sensor value(s)
29       if Touch then
30          DIO.Put_Line ("Button is pressed!");
31       end if;
32       DIO.Put ("Count: "); DIO.Put_Line (Count'Img);
33
34       -- wait for some time
35       Arduino.C_Delay (1000);
36
37    end loop;
38 end EvShield_Touch_Sensor;
```

Listado 7: Programa EVShield_Touch_Sensor desarrollado tras la readaptación de los conectores.

En el Listado 6 se observa como se tiene que crear el objeto del EVShield en las lineas 16-17 y como en la 25 se crea el objeto del sensor. Ya en la sección begin de la aplicación se encuentra la inicialización tanto del protocolo I2C utilizado en la linea 28 (se utiliza el protocolo Hardware como recomienda el desarrollador) y la inicialización del sensor de contacto en las lineas 30-31. Todo esto es suplido en el Listado 7 con código implementado con los nuevos paquetes genéricos en la linea 19.

La otra diferencia encontrada entre estos códigos es la forma de llamar a la función que obtienen la cuenta de veces que ha sido el sensor pulsado y la función que obtiene si el sensor esta siendo presionado en ese instante de tiempo. En el código anterior a la adaptación (Listado 6), estas se encuentran en las lineas 34 y 35, mientras que en el nuevo código (Listado 7) son las lineas 25 y 26.

Como puede apreciarse comparando los listados anteriores, la utilización de los conectores adaptados permite un código mucho más sencillo y legible, el cuál facilita su uso al programador, lo cuál era uno de los objetivos principales del proyecto debido al enfoque educacional que se le da a esta plataforma.

Otros ejemplos de mejora en otros programas desarrollados son las siguientes mostrados, los cuales pertenecen a extractos de un programa de prueba del sensor NXTColor motores (Listado 8) y otro del programa Motor_Rotations (Listado 9).

```
-- Create shield object
EVShield : aliased EVShield_h.Class_EVShield.EVShield :=
    EVShield_h.Class_EVShield.New_EVShield (16#34#,16#36#);

-- Declare our sensor for use in this program
Color_Sensor : aliased EVs_NXTColor_h.Class_EVs_NXTColor.EVs_NXTColor;

aux := EVs_NXTColor_h.Class_EVs_NXTColor.init(Color_Sensor'Access,
    EVShield'Access, SH_BAS1);
aux := EVs_NXTColor_h.Class_EVs_NXTColor.setType(Color_Sensor'Access,
    EVShield_h.SH_Type_COLORNONE);

-- read the value from color sensor
value := EVs_NXTColor_h.Class_EVs_NXTColor.readValue(Color_Sensor'Access);
```

Listado 8: Fragmento del programa NXTColor_Color desarrollado con los conectores autogenerados.

```
-- Create shield object
EVShield : aliased EVShield_h.Class_EVShield.EVShield :=
    EVShield_h.Class_EVShield.New_EVShield (16#34#,16#36#);

encoder2 := EVShield_h.Class_EVShieldBank.motorGetEncoderPosition
    (EVShield.bank_a'Access, SH_Motor_2);

output := EVShield_h.Class_EVShieldBank.motorRunRotations(EVShield.bank_a'Access,
                                EVShield_h.SH_Motor_1,
                                EVShield_h.SH_Direction_Forward,
                                EVShield_h.SH_Speed_Medium,
                                2,
                                EVShield_h.SH_Completion_Wait_For,
                                EVShield_h.SH_Next_Action_BrakeHold);
```

Listado 9: Fragmento del programa Motor_Rotations desarrollado con los conectores autogenerados.

Frente al código mostrado en el Listado 10 y el Listado 11 que ha sido desarrollado usando los conectores adaptados.

```
package Color_sensor is new Arduino.EVShield.NXTColor.Color (Arduino.EVShield.SH_BAS1);

-- read the value from color sensor
value := Color_sensor.Read_Color;
```

Listado 10: Fragmento del programa NXTColor_Color desarrollado con los adaptados.

```
-- Bank A Motor 1
package BAM1 is new Arduino.EVShield.Motors.Motor (Arduino.EVShield.Motors.SH_BAM1);
-- Bank A Motor 2
package BAM2 is new Arduino.EVShield.Motors.Motor (Arduino.EVShield.Motors.SH_BAM2);

encoder2 := BAM2.Motor_Get_Encoder_Position;

output := BAM1.Motor_Run_Rotations(BAM1.SH_Direction_Reverse,
                                   BAM1.SH_Speed_Medium,
                                   2,
                                   BAM1.SH_Completion_Wait_For,
                                   BAM1.SH_Next_Action_BrakeHold);
```

Listado 11: Fragmento del programa Motor_Rotations desarrollado con los conectores adaptados.

4.6. Sensores LEGO soportados.

Como se ha comentado en la Sección 3.1, la librería EVShield soporta un gran número de sensores y actuadores de las distintas versiones de los LEGO Mindstorm y en este proyecto se utilizarán varios de los más populares.

Para cada sensor/motor se ha generado un paquete siguiendo la estrategia de uso de genéricos descrita en el apartado anterior (Sección 4.5).

4.6.1. EV3Touch.

El sensor de contacto EV3 es una simple pero funcional herramienta que sirve para detectar tanto la presión del botón como la liberación.[22]

En el paquete genérico Arduino.EVShield.EV3Touch se realiza la conexión con la librería EVShield con la que se consigue dar la funcionalidad correcta al sensor.

Figura 10: Sensor EV3Touch.

4.6.2. EV3Ultrasonic.

El sensor de ultrasonidos EV3 emite ondas de sonido y detecta sus ecos para medir así la distancia a la que se encuentra un objeto. Mide distancias entre 1 y 250 cm con una precisión de +/- 1 cm.[24]

En el paquete genérico Arduino.EVShield.EV3Ultrasonic se realiza la conexión con la librería EVShield con la que se consigue dar la funcionalidad correcta al sensor.

Figura 11: Sensor EV3Ultrasonic.

4.6.3. EV3Color.

El sensor EV3Color es un sensor digital capaz de reconocer 8 colores distintos, así como también servir de sensor de luz siendo capaz de detectar cambios en las intensidades.[23]

En el paquete genérico Arduino.EVShield.EV3Color se realiza la conexión con la librería EVShield con la que se consigue dar la funcionalidad correcta al sensor.

Figura 12: Sensor EV3Color.

4.6.4. NXTColor.

Versión anterior al sensor de color EV3. Las diferencias con este otro es que la versión NXT detecta menos colores y no tiene la capacidad de medir la luz reflejada, de mientras, este incluye la funcionalidad de mostrar luz, es decir, funcionar como una bombilla.[20]

En el paquete genérico Arduino.EVShield.NXTColor se realiza la conexión con la librería EVShield con la que se consigue dar la funcionalidad correcta al sensor.

Figura 13: Sensor NXTColor.

4.6.5. Motores.

Se disponen de 3 versiones de servo-motores, los cuales son:

- Motor grande EV3. Potente motor con un tacómetro con una precisión de 1 grado. Es capaz de funcionar a entre 160 y 170 rpm, con una fuerza de torsión en movimiento de 20 N/cm y de 40 N/cm en parado. [25]

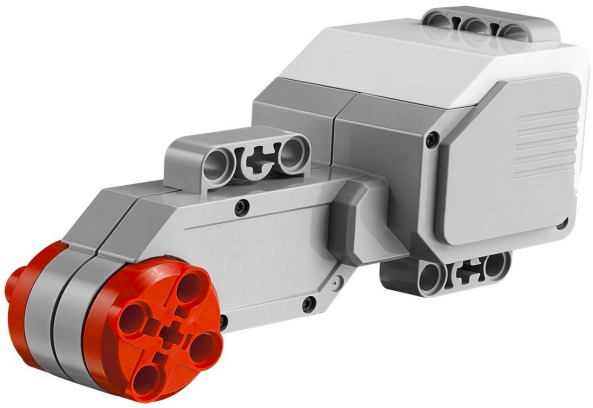

Figura 14: Motor Grande EV3.

- Motor mediano EV3. Motor que es capaz de alcanzar una velocidad de entre 240 y 250 rpm. De mientras, la fuerza de torsión que es capaz de mostrar es menor que la del anterior, con 8 N/cm en movimiento y 12 N/cm en estático.[26]

Figura 15: Motor Mediano EV3.

- Motor NXT. Servo motor de la serie NXT con velocidades máximas y fuerzas de torsión menores a los dos anteriores.[21]

Figura 16: Motor NXT.

En el paquete genérico Arduino.EVShield.Motors se realiza la conexión con la librería EVShield con la que se consigue dar la funcionalidad correcta a los motores.

5. Evaluación y pruebas.

Se han realizado una serie de aplicaciones con el objetivo de comprobar el correcto funcionamiento tanto del escudo EVShield como de los distintos sensores de los que se dispone para realizar el proyecto. Estas pruebas constituyen, además, ejemplos de uso de los distintos dispositivos.

5.1. EVShield.

5.1.1. Blinkers.

El programa denominado blinkers permite probar el correcto funcionamiento de los botones y leds integrados en el propio escudo EVShield. Su funcionamiento es bastante simple, para empezar se debe pulsar el botón GO y tras esto se podrá pulsar tanto el botón derecho como el izquierdo (parte superior derecha) y se encenderán los leds respectivos al banco del botón pulsado. Los leds de cada banco se encuentran en las esquinas inferiores.

Los leds pueden ser configurados para mostrar cualquier color del espectro RGB.

5.1.2. Print_Info.

Este programa es utilizado como comprobación de las características de los bancos del escudo. Tras pulsar el botón GO, imprimirá por pantalla para cada banco la versión del Firmware, el Id del vendedor del producto (MindSensors), el Id del dispositivo y qué banco es.

5.2. EV3Touch.

5.2.1. Detección de pulsado del botón.

Este programa permite probar el correcto funcionamiento del sensor. Detectará tanto si el botón está pulsado en un instante de tiempo con la función Is_Pressed, como también, cuantas veces ha sido pulsado con la función Get_Bump_Count.

También se podrá tener la posibilidad de hacer un reset del contador de pulsaciones con la función Reset_Bump_Count.

5.3. EV3Ultrasonic.

5.3.1. Detección de distancia.

Este programa sirve para averiguar la distancia a la que se encuentra un objeto. El sensor trabaja en un rango entre 1 y 250 cm.

Este programa esta por defecto para conseguir la distancia en centímetros, pero modificando la inicialización del modo del sensor se puede configurar para que muestre resultados en pulgadas.

5.3.2. Detección de presencia.

Este programa podrá detectar la presencia de ondas de ultrasonidos provenientes de otros sensores de distancia.

5.4. EV3Color.

5.4.1. Detección de luz ambiente.

Con este programa se conseguirá detectar la cantidad de luz que hay en un lugar, consiguiendo valores que indican desde la oscuridad hasta una gran claridad.

5.4.2. Detección de luz reflejada.

Este programa medirá la cantidad de luz roja reflejada.

5.4.3. Detección de color.

El sensor con este programa es capaz de diferenciar entre 7 colores (azul, verde, negro, amarillo, rojo, blanco y marrón) y la ausencia de color.

5.5. NXTColor.

5.5.1. Detección de luz ambiente.

Al igual que el sensor de color EV3, este sensor será capaz de medir la cantidad de luz.

5.5.2. Muestra de colores.

El sensor de color NXT es capaz de mostrar luz de color rojo, verde, azul y blanco. Estos haces de luz se irán sucediendo de forma periódica en este programa.

5.5.3. Detección de color.

Este sensor será capaz de diferenciar entre los colores: azul, verde, rojo, amarillo, blanco y negro.

5.6. Motores.

5.6.1. Rotaciones.

Este programa permite manejar entre 1 y 4 motores los cuales darán de uno en uno dos rotaciones con velocidad positiva y dos con velocidad negativa.

5.6.2. Tacómetro.

Este programa medirá cuantos grados mueves los ejes de rotación de los motores que conectes al escudo.

5.6.3. Motor con sensor de contacto.

En este programa se unen dos piezas, un sensor de contacto y un motor. Con esto se consigue un sistema de encendido y apagado del motor. Mantener el botón del sensor de contacto pulsado hará funcionar al motor. Una vez soltado el botón, el motor parará.

6. Demostradores.

Se han desarrollado 3 aplicaciones mas complejas que los ejemplos anteriores con las cuales poder apreciar el verdadero potencial de la plataforma desarrollada.

Para la realización de estos demostradores se ha seguido una estrategia de programación multitarea utilizando las tareas Ada proporcionadas por el sistema operativo M2OS.

Las tareas Ada se pueden ver como hilos de control que se encargan de realizar una tarea, es decir, similares a los threads POSIX. [3]

6.1. Conducción autónoma.

La conducción autónoma de vehículos es una de las tecnologías que mas se están desarrollando en los últimos años, algunas empresas como puede ser Tesla son las mayores referentes en este ámbito. Esta demostración, con el uso de detección de distancia y control de motores pretende dar una base sencilla sobre la que podrían desarrollarse aplicaciones más complejas de conducción autónoma.

Para esta demostración hacemos uso de dos motores NXT, un sensor de contacto, dos sensores de ultrasonidos (cada uno apuntando en una dirección ligeramente diferente) y el escudo EVShield con la placa Arduino Uno. El robot construido se muestra en la Figura 17.

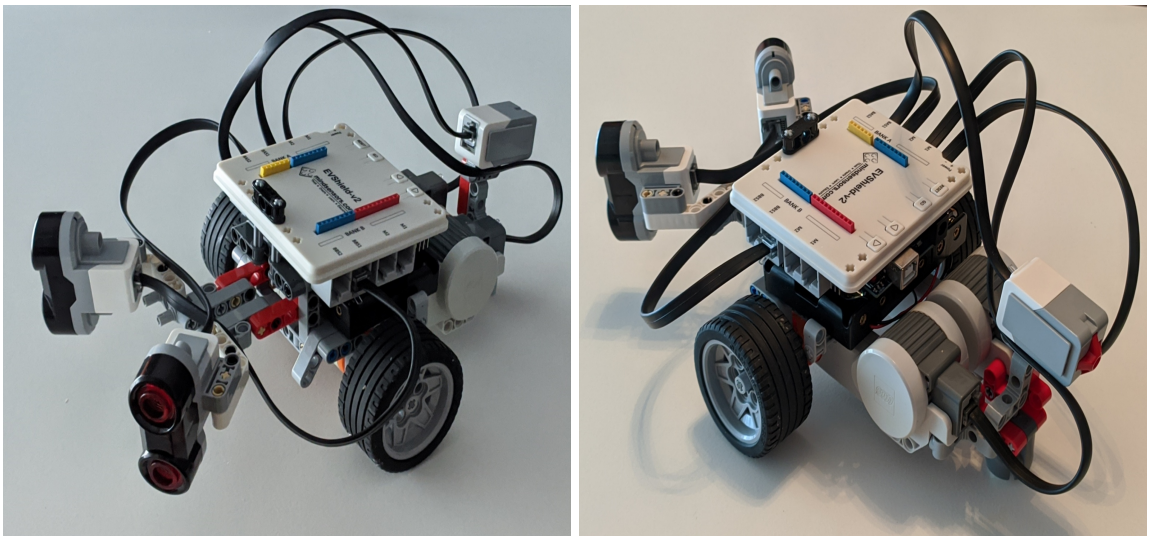

Figura 17: Robot para conducción autónoma.

La jerarquía de tareas utilizada para el desarrollo de la demostración es la encontrada en la Figura 18. Se posee una tarea principal que es denominada como tarea *Main* la cuál se encarga de planificar las dos tareas trabajadoras, en la parte derecha se encuentra la tarea *Measures*, esta se ocupa de obtener las medidas de los sensores de distancia y proporcionar ese dato a la otra tarea. La tarea denominada *Control* es la encarga de, con los datos obtenidos, guiar al robot.

El periodo propuesto para ambas tareas es de 250 ms.

Para el guiado del vehículo se ha seguido una lógica bastante sencilla con la que se va comprobando las distancias obtenidas por ambos sensores y se evalúan para realizar el movimiento mas adecuado.

Pulsando el sensor de contacto se podrá parar o reanudar la demostración en cualquier momento.

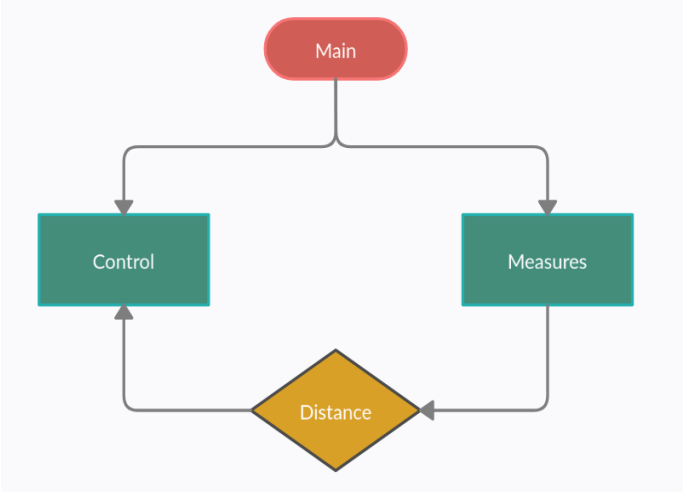

Figura 18: Diagrama de tareas para la demostración de conducción autónoma.

6.2. Robot sigue luz.

Con esta demostración se ha buscado que nuestro dispositivo sea capaz de seguir una fuente de luz. A través de dos sensores de color configurados para medir la intensidad de luz, comprobará por cual recibe más cantidad de lummens y lo seguirá.

La arquitectura del robot utilizado para esta demostración es similar al utilizado para la conducción autónoma con la diferencia de utilizar un sensor de color EV3 y un sensor de color NXT en vez de los dos sensores de ultrasonidos EV3. El robot construido se muestra en la Figura 19.

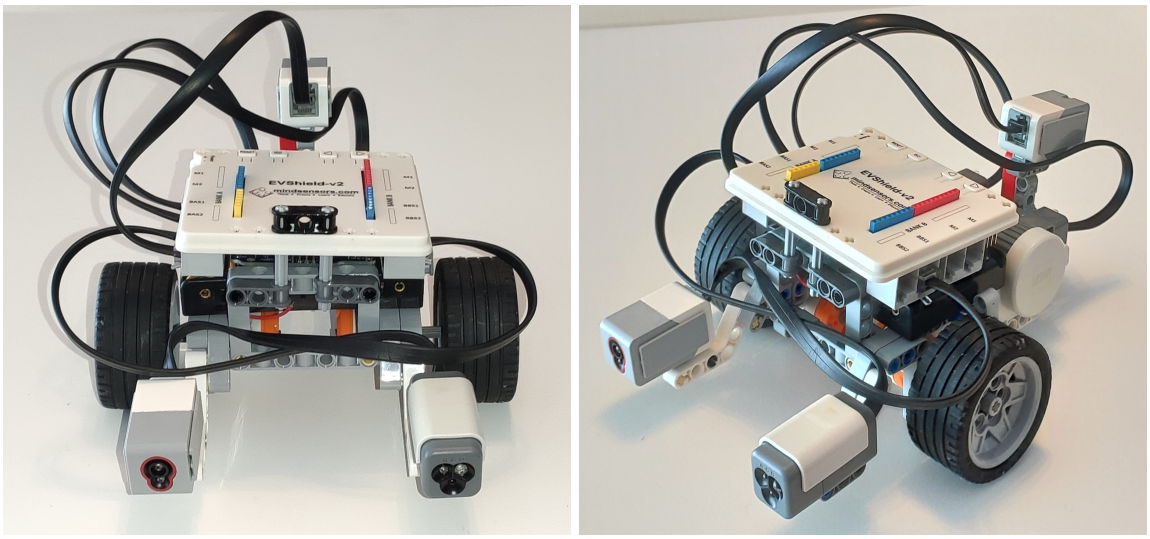

Figura 19: Robot sigue luz.

La jerarquía de tareas del robot que sigue la luz, la cuál es mostrada en la Figura 20, es bastante similar a la utilizada para la demostración anterior, con la diferencia de que en este caso el dato compartido entre las tareas es la intensidad de luz recibida por los sensores.

Para esta demostración se utiliza un periodo de 250 ms.

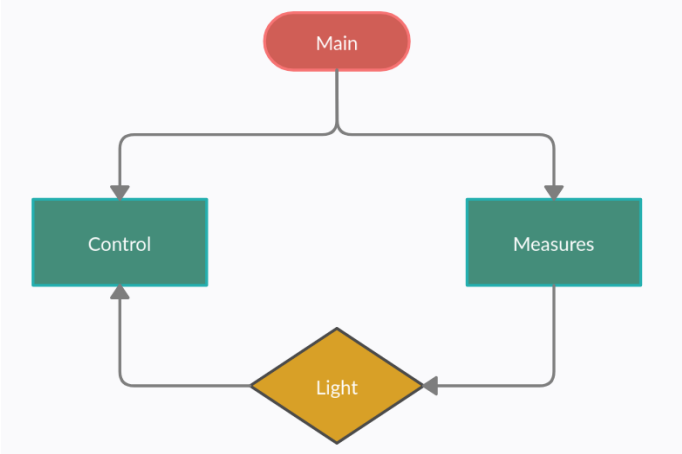

Figura 20: Diagrama de tareas para la demostración del robot que sigue la luz.

6.3. Robot sigue lineas.

Con esta demostración, nuestro robot tendrá la habilidad de seguir un circuito marcado por una linea negra. Se dispone de dos sensores de color los cuales deberán estar separados adecuadamente ya que entre ellos tese encuentra la linea que marca el circuito. El robot empezará a acelerar comprobando que ninguno de los dos sensores pisa la linea. En caso positivo se llevarán las acciones pertinentes para volver a poner al robot en posición correcta.

Para este robot se utilizan los mismos elementos que para el robot que sigue la luz, es decir, dos motores NXT, un sensor de contacto con el que parar o encender el robot, dos sensores de color, uno de la serie EV3 y otro de la serie NXT y por último, como es lógico, el escudo EVShield y la placa Arduino. El robot construido se muestra en la Figura 21.

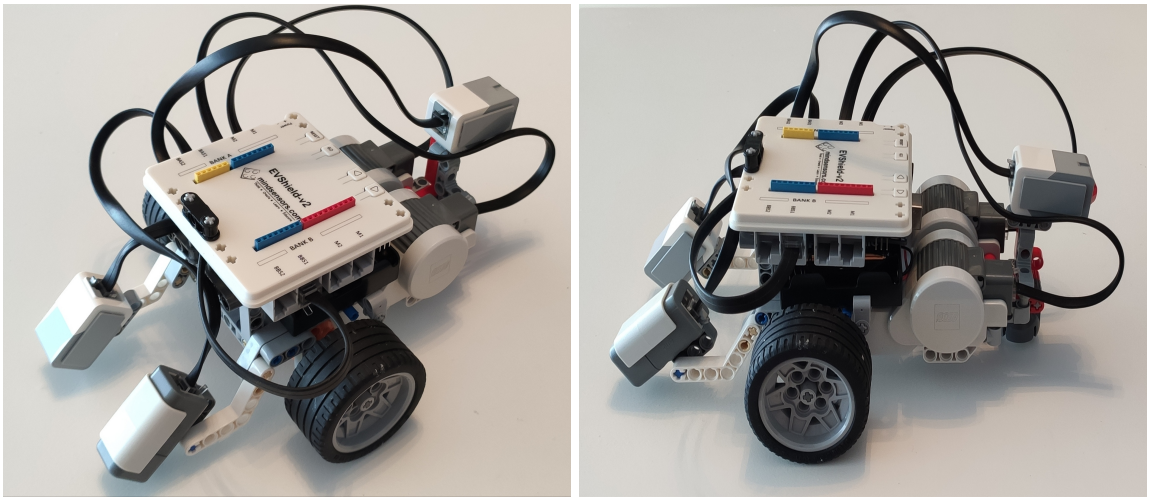

Figura 21: Robot sigue lineas.

Por último, esta demostración sigue la misma estructura de tareas que las dos demostraciones anteriores (Figura 22) pero con la diferencia de que en este caso el dato compartido entre las tareas es el color obtenido por los sensores. Por otra parte, en esta demostración las tareas trabajadoras tienen un periodo de 200 ms.

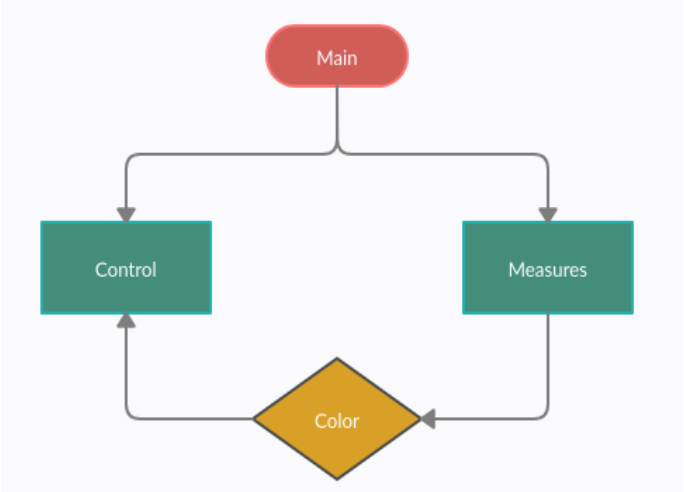

Figura 22: Diagrama de tareas para la demostración del robot sigue lineas.

7. Conclusiones.

Este proyecto tenía el objetivo de realizar un marco de desarrollo para aplicaciones Ada con actuadores Lego en Arduino. Para ello nos hemos apoyado en el escudo EVShield y el sistema operativo M2OS.

Los microcontroladores son unos dispositivos muy utilizados tanto a nivel industrial, como para beneficio personal y cada vez mas, a nivel educativo ya que otorgan un amplio rango de posibilidades con un presupuesto relativamente bajo. Este es uno de los motivos por los cuales este proyecto ha sido muy interesante debido a que he podido experimentar con el mundo de la robótica y dejar una plataforma muy interesante para futuros alumnos.

Se ha conseguido realizar una implementación totalmente funcional, la cuál pasará a ser parte de la distribución M2OS y con un gran potencial de ser una plataforma donde futuros alumnos de la Universidad de Cantabria puedan desarrollar prácticas con las que mejorar sus conocimientos en robótica y aplicaciones concurrentes.

Para dar una mayor facilidad de uso, además, se ha realizado una readaptación de los conectores siguiendo una estrategia de paquetes genéricos con la que se ha conseguido un código más claro y legible.

Este trabajo, además, ha permitido continuar con el desarrollo de los conocimientos adquiridos a lo largo del grado den Ingeniería Informática, en especial de asignaturas como pueden ser Sistemas de Tiempo Real o Sistemas Operativos Avanzados, ambas de la mención de computadores.

Ada es un lenguaje que aunque todavía no ha visto despuntar su popularidad para un usuario normal, su uso es bastante extenso en áreas como la ingeniería aeroespacial o aeronáutica.

Uno de los problemas que conlleva este lenguaje es la escasa cantidad de ejemplos encontrados por la red, sobre todo en relación a la conexión del lenguaje con otros. Esta parte del proyecto, al ser una situación novedosa y ser el núcleo del desarrollo del marco, hay que ponerla especial atención.

8. Trabajos futuros.

Expongo en las siguiente lineas, unas propuestas de mejora para el proyecto realizado.

En el desarrollo de las demostraciones se encontró un fallo con el banco B. Si hay varios sensores conectados de forma simultanea en ambos bancos y se conecta un motor en el banco B, los sensores conectados en este banco dejan de funcionar.

Para intentar arreglar este error se procedió a implementar un depurador. Se encontró un proyecto desarrollado por Jan Dolinay, el cuál permite depurar códigos Arduino con sistema basado en el depurador GDB.[12]

Aunque este depurador, funcionaba correctamente con aplicaciones Arduino construidas con el IDE oficial. Sin embargo, al tratar de usarlo con programas que utilizaban M2OS obteníamos errores de compatibilidad, por lo que se decidió dejar de lado la inclusión del depurador.

La inclusión tanto del depurador como la corrección del error encontrado serán las lineas a seguir para futuros proyectos relacionados con este marco de desarrollo.

Otra linea de trabajo será la inclusión en M2OS de más sensores y actuadores LEGO compatibles con el escudo EVShield.

Bibliografía.

[1] AdaCore. *Ada, The language for safe, secure and reliable Software*. URL: https://www.adacore.com/about-ada.

[2] AdaCore. *GNAT Studio. The simple powerful IDE*. URL: https://www.adacore.com/gnatpro/toolsuite/gnatstudio.

[3] AdaCore. *Tasking*. URL: https://learn.adacore.com/courses/intro-to-ada/chapters/tasking.html.

[4] Mario Aldea Rivas y Hector Perez Tijero. "Leveraging real-time and multitasking Ada capabilities to small microcontrollers". En: *Journal of Systems Architecture* 94 (2019), págs. 32-41. ISSN: 1383-7621. DOI: https://doi.org/10.1016/j.sysarc.2019.02.015. URL: https://www.sciencedirect.com/science/article/pii/S1383762118302212.

[5] Arduino. *Arduino IDE 1.8.15 Download*. URL: https://www.arduino.cc/en/software.

[6] Arduino. *Install the Arduino Software (IDE) on Linux*. URL: https://www.arduino.cc/en/Guide/Linux.

[7] Arduino. *Software de Arduinor*. URL: https://arduino.cl/programacion/.

[8] Arduino. *What is Arduino?* URL: https://www.arduino.cc/en/Guide/Introduction.

[9] Aprendiendo Arduino. *I2C*. URL: https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/.

[10] Universidad de Cantabria. *M2OS. RTOS with simple tasking support for small microcontrollers.* URL: https://m2os.unican.es/.

[11] DescubreArduino. *Arduino Uno, partes, componentes, para qué sirve y donde comprar*. URL: https://descubrearduino.com/arduino-uno/.

[12] Jan Dolinay. *Source level debugger for Arduino*. URL: https://github.com/jdolinay/avr_debug.

[13] Ada Europe. *Ada Reference Manual - Interfacing Pragmas*. URL: https://www.adaic.org/resources/add_content/standards/05aarm/html/AA-B-1.html.

[14] Ada Europe. *Ada Reference Manual - The Package Interfaces*. URL: https://www.adaic.org/resources/add_content/standards/05aarm/html/AA-B-2.html.

[15] Mindsensors - EVSHield. *EVSHield Library*. URL: https://github.com/mindsensors/EVShield.

[16] Mindsensors - EVSHield. *EVSHield Library class index*. URL: http://www.mindsensors.com/reference/EVShield/html/index.html.

[17] Mindsensors - EVShield. *EVShield Library Tutorial*. URL: http://www.mindsensors.com/pdfs/EVShield-Library-Tutorial.pdf.

[18] MicroChip. *AVR® and Arm® Toolchains (C Compilers)*. URL: https://www.microchip.com/en-us/development-tools-tools-and-software/gcc-compilers-avr-and-arm.

[19] Mindsensors. *Mindsensors*. URL: http://www.mindsensors.com/.

[20] LEGO MINDSTORMS. *LEGO Education NXT Colour Sensor*. URL: https://www.electricbricks.com/lego-education-mindstorms-sensor-nxt-colour-sensor-p-4580.html.

[21] LEGO MINDSTORMS. *LEGO Education NXT Interactive Servo Motor*. URL: https://www.electricbricks.com/lego-education-mindstorms-nxt-nxt-interactive-servo-motor-p-252.html.

[22] LEGO MINDSTORMS. *LEGO MINDSTORMS Education EV3 Touch Sensor*. URL: https://education.lego.com/es-es/products/lego-mindstorms-education-ev3-touch-sensor/45507.

[23] LEGO MINDSTORMS. *Sensor de color EV3*. URL: https://www.lego.com/es-es/product/ev3-color-sensor-45506.

[24] LEGO MINDSTORMS. *Sensor Ultrasónico EV3*. URL: https://www.lego.com/es-es/product/ev3-ultrasonic-sensor-45504.

[25] LEGO MINDSTORMS. *Servomotor Grande EV3*. URL: https://www.lego.com/es-es/product/ev3-large-servo-motor-45502.

[26] LEGO MINDSTORMS. *Servomotor Mediano EV3*. URL: https://www.lego.com/es-es/product/ev3-medium-servo-motor-45503.

[27] WIKILIBROS. *Programación en Ada/Unidades genéricas*. URL: https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Ada/Unidades_gen%C3%A9ricas.

[28] Xataka. *Qué es Arduino, cómo funciona y qué puedes hacer con uno*. URL: https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno.

A. Instalación del IDE de Arduino.

Para la instalación del IDE de Arduino se seguirán los siguientes pasos:

1. En la pagina principal de Arduino se pueden encontrar los paquetes necesarios para la instalación, mas concretamente en la pestaña Software. En ella se encuentran diversos paquetes para diferentes sistemas operativos, en nuestro caso se debe trabajar sobre Linux.[5]

2. Una vez elegido el paquete adecuado para nuestra arquitectura lo descargaremos y descomprimiremos.[6]

   ```
   $ tar -xvf arduino-1.8.13-linux64.tar.xz
   ```

3. El último paso será acceder a la carpeta descomprimida y ejecutar el script de instalación install.sh

   ```
   $ cd arduino-1.8.13/
   $ ./install.sh
   ```

Un error conocido tras la instalación del IDE es que tras elegir correctamente la placa utilizada y el serial port adecuado, al cargar un sketch recibir el fallo: Error opening serial port ...

Este fallo es debido a una incorrecta configuración de los permisos de los puertos serie, para ellos se debe comprobar nuestro caso y corregirlo. Para ello se puede ejecutar el siguiente comando.

```
$ ls -l /dev/ttyACM*
```

Y se obtiene una salida similar a esta:

```
crw-rw---- 1 root dialout 188, 0 5 apr 23.01 ttyACM0
```

El 0 del final del output del comando puede que sea un número distinto. El dato importante es "dialout", que es el grupo al que pertenece el puerto. A este se debe incluir nuestro usuario con el siguiente comando.

```
$ sudo usermod -a -G dialout <username>
```

Donde username será nuestro usuario. Para que este cambio haga efecto será necesario hacer un salir y entrar de la sesión de nuestro usuario y con esto el error debería haber desaparecido.

B. Instalación de la librería EVShield.

Una vez ya instalado el IDE de Arduino y conseguido su correcto funcionamiento se procederá a la instalación de la librería EVShield. Para ello se deben seguir los siguientes pasos:

1. Se puede encontrar la librería en un repositorio github publico por lo que se utiliza el siguiente comando para descargarla.[15]

   ```
   $ git clone https://github.com/mindsensors/EVShield.git
   ```

2. La librería debe estar situada dentro de la carpeta libraries de Arduino, la cuál debería tener una ruta similar a /Arduino/libraries

   ```
   $ mv EVShield ~/Arduino/libraries/
   ```